A Demonstration of the Application of Attributed Task-Graphs for Distributed Task Execution in Mobile Ad Hoc Networks[1]

P. Basu, W. Ke, S. Abu Ayyash, and T.D.C. Little

Department of Electrical and Computer Engineering
Boston University, Boston, Massachusetts 02215, USA
(617) 353-9877
tdcl@bu.edu

MCL Technical Report No. 10-16-2003

1 Introduction

Pervasive networks can enable a broad range of distributed applications that need exchange of information between multiple devices. We developed a set of techniques [3, 1] for modeling and intelligent execution of such distributed applications (or tasks) on a network of possibly mobile devices. In [2, 3] we introduced the abstraction of a *Task Graph* (TG) for logically representing higher-level tasks that a user may want to perform, in terms of smaller sub-tasks. It is a graph composed of *nodes* and *edges*, where the nodes represent the *classes* of devices/services (e.g., Printer, Photocopier, Digital Picture Frame etc.) needed for processing data related to the task while the edges represent necessary associations between different nodes for performing the task. Our technique takes into account the nature and requirements of a given application task for making decisions about the set of devices that the application should run on.

In this demo we use a network of wireless devices operating in ad hoc mode. Our protocols will dynamically assign components of an application to suitable nodes of the MANET, and will adapt its execution to changes in the underlying network, such as node mobility and failures. Our techniques provide resiliency and flexibility in adapting distributed applications in an inherently unpredictable and unreliable distributed system.

We have reported simulation results of the application of our techniques in various

[1] *Demo Session, ACM MOBICOM03*, San Diego, CA, September 2003. This material is based upon work supported by the National Science Foundation under Grant No. ANI-007384. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

places [1, 4, 3]. As a proof of concept, we have recently built a testbed with added software capabilities which we briefly describe in this paper. For the purposes of demonstrating the outcome of our research activities, we focus on a polling application in which an initiating node maps the application onto the MANET based on a TG representation. When a user wishes to request a poll to be executed, she specifies the application's logical structure, requirements and possibly code necessary via a concise data dependency structure we refer to as *Data-Flow Tuples*. Henceforth, suitable devices are seamlessly selected (in other words, *instantiated*) at runtime, and are made to communicate with one another according to the specifications of the TG (e.g., required number of polling samples, required location and type of polled devices etc.). In particular, for each class of device in TG, one suitable *instance* is chosen to participate in the task execution. In the event of a disruption in the application owing to failures in any component in the network, our algorithm quickly and seamlessly adapts to the situation and dynamically *re-instantiates* the affected parts of the TG, if possible.

Through the polling application demo we can demonstrate the following processes:

1. Parsing of user supplied data-flow tuples (application description file) and the instantiation of a task graph on nodes of a MANET in both static and mobile scenarios.
2. Timely delivery of task specific data to the currently chosen recipients.
3. Timely detection of disconnections between two participating hosts as a result of failure, or changes in the underlying MANET topology at any time, even during the TG instantiation phase.
4. Rapid re-instantiation of the task graph with mostly local coordination, and resumption of task data delivery.

The following sections describe the demo in more detail, in addition to providing some background material on the design and the implementation of our protocols.

2 A Brief Description of the Software Architecture

We implemented the protocol finite state machine (FSM) outlined in [4, 1] in the application layer of the networking stack of each host in the testbed, thus requiring no change to the existing Linux kernel. Linux socket APIs were used to facilitate communication between

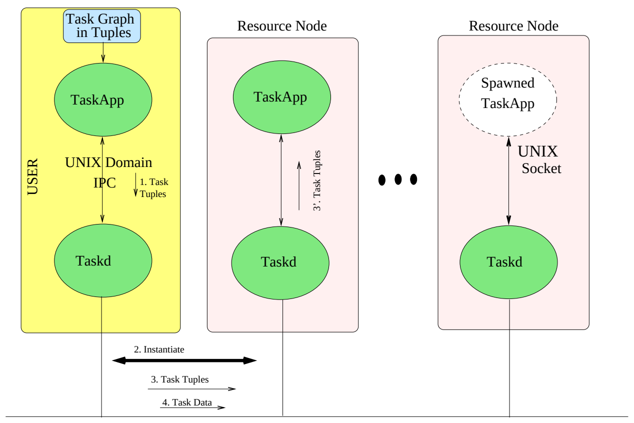


Figure 1: Software Architecture

nodes in the testbed for exchange of both protocol and data messages. We implemented the task graph instantiation protocol module as a daemon process *taskd* which runs on each node in the testbed.

The module corresponding to the distributed task initiated by the user which requires services and resources from other nodes in the MANET is referred to as a *taskapp*. A *taskapp* possesses knowledge of the corresponding TG and uses Linux IPC to instruct its local *taskd* to begin instantiation of the TG.

Although *taskd* is an application layer daemon, it must be logically separated from a *taskapp*. Each *taskapp* needs a *taskd* associated with it. While the latter handles instantiation, failure and disconnection detection, and re-instantiations, the former handles application level details such as processing the data sent by a remote service, reaction to failures and delays at the *taskd* layer, and buffering of Application Data Units (ADUs) for possible retransmission at a later stage. Figure 1 describes the logical separation between the two processes *taskapp* and *taskapp*.

3 Description of Demo

To validate the proposed concepts briefly described above, we developed a prototype testbed on which we demonstrate a simple polling application where a user requests to poll k sensors with desired attributes, e.g., location and sensor type. We break down the polling task into two layers of polling managers and actual sensors. Our distributed protocol utilizes a greedy algorithm that uses local search for discovering suitable candidate nodes with matching attributes while satisfying the constraints imposed by the polling task graph structure.

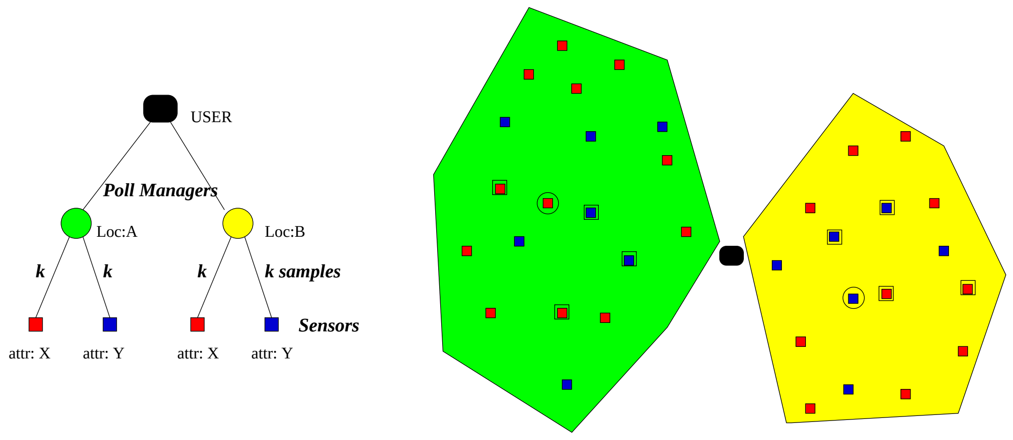


Figure 2: A Polling Task Graph and its Instantiation

Figure 2 illustrates the logical task graph representation of the polling application and a physical instance which emphasizes the nodes chosen to realize it – squares bordered with circles signify poll managers and those bordered with larger squares signify selected pollees.

Initially, the user device (U) will broadcast a query packet (with TTL scoping) requesting devices in locations loc:A and loc:B to take up the roles of polling managers for those locations respectively. After a brief protocol handshake, U will instantiate TG-nodes A and B with suitable devices in respective locations. The instantiated polling managers at loc:A and loc:B will in turn instantiate k sensors satisfying the location and type attributes specified in the polling task graph.

The proposed protocol is also capable of recovering from node and route failures inherent in a MANET by scalable mechanisms of disconnection detection, re-instantiation, and TG-patching [1, 4]. These recovery steps are necessary in situations where polling sessions are long and interactive. For example, if an instantiated polling manager in loc:B fails or moves away, U detects this disconnection and then discovers and re-instantiates another device in loc:B to perform the role of the previous polling manager. Associations between the new polling manager and older pollees are then established along with relevant application state. All these steps demonstrating the adaptation of the polling application will be highlighted in the demo.

Metrics We measure the success of our distributed instantiation algorithm by certain metrics such as average time for instantiation, average dilation (a measure of hop distances between relevant instantiated nodes), frequency of disruption of tasks due to mobility or route failures, and average effective throughput and application data delay that are experienced after task data transmission begins.

3.1 Demo Equipment

Hardware: Our testbed is comprised of 15 computer components which include 3 desktop computers, 7 portable notebooks and 5 handheld computers. Each of these machines undertakes a certain role in the polling demo – some devices will act as Temperature and Pressure sensors, and others with greater computational resources and battery power will take up the roles of poll managers. However, given possible limitations of demo space and facilities, a portable subset of these machines will be involved in the actual demonstration.

Operating Systems: All hosts in the experimental testbed execute different versions of the Linux 2.4 kernel.

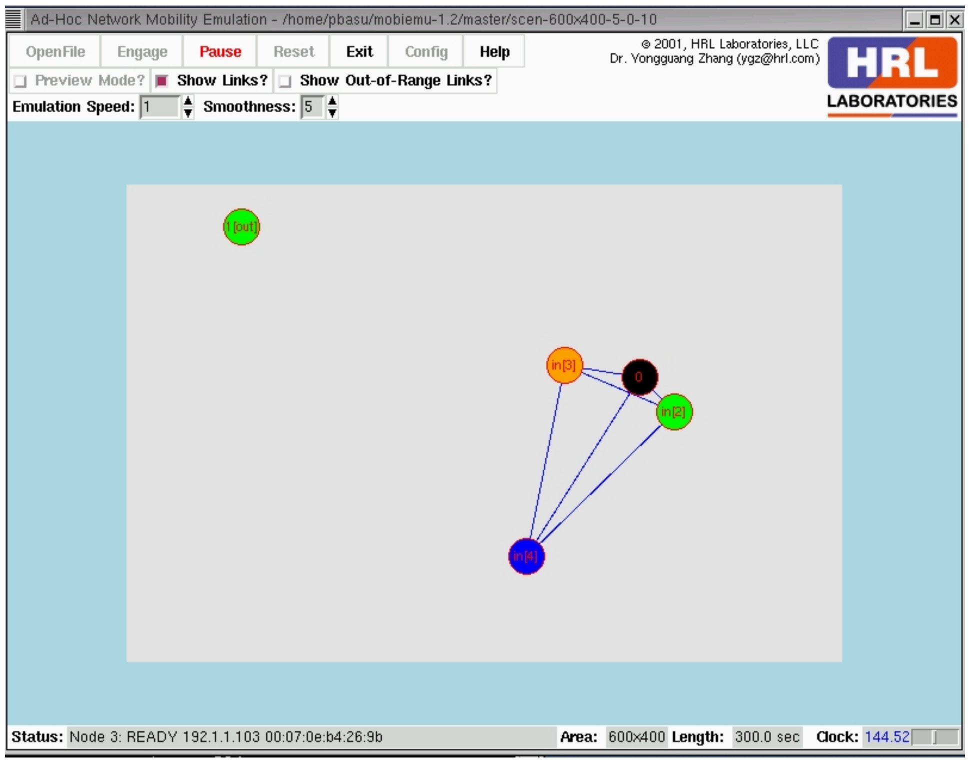
Networking Hardware and Software: Each of the above devices is equipped with Cisco Systems 350 Series Aironet wireless PCMCIA cards. The radios in the wireless cards operate in the unlicensed ISM 2.4GHz frequency band using Direct Sequence Spread Spectrum (DSSS) at a data rate of 11Mbps. All wireless cards support the IEEE 802.11b standard and will be operated in ad hoc mode in this demo. The radios will tuned to the same frequency (which should be different from the ones used by other demos in the room) and their transmit power will be set to the minimum achievable, i.e., 1 mW.

Routing Protocol Support: We utilized a public domain implementation of a popular MANET routing protocol for achieving multihop routing, namely OLSR (from INRIA, France) [5]. The above implementation resides in the user space of the Linux OS. We were able to port this to the StrongARM platform (iPAQ) easily.

The MobiEmu Toolkit: Since forming a multihop network on the demo floor is difficult even at 1mW xmit power, we will use the MobiEmu toolkit [6] which allows one to emulate a multihop MANET in the absence of multihop paths and real mobility. Each host that wants to participate in the MANET runs a MobiEmu slave. An additional host runs a MobiEmu master which is aware of the entire time-varying mobility pattern that we want to emulate using the slave hosts. The master and all slave hosts communicate with each other via IP multicast over either the same wireless channel or preferably over a wired network. (This is to mitigate the effect of loss of control traffic information in a shared wireless network.) Complicated mobility patterns can be emulated while using real hardware and real routing protocols without physically moving the hosts. Hence MobiEmu serves as an effective tool for testing the feasibility of our protocols on real hardware under random mobility patterns.

Figure 3 shows an image of the MobiEmu visualization screen at a particular time instant before which a four node task graph has been instantiated and then re-instantiated. Of the five nodes in the MANET, device-0 belongs to the user, device-1 and device-2 possess identical attributes, and device-3 and device-4 possess distinct attributes. Originally devices 1, 3, and 4 were instantiated (depicted by `[in]`); at the time of screen capture, device-1 has moved out of range and hence has been uninstantiated (depicted by `[out]`). In its place, the corresponding TG node has been instantiated with device-2.

Layout of the demo: The layout of the demo can be simple where all wireless devices can be scattered on a single lab-size table. Viewers can monitor the state of nodes during the progression of the polling process on each machine. They can graphically monitor the instantiation, mobility, disconnection and recovery (re-instantiation) on the MobiEmu *master* machine (desktop/notebook), see Figure 3.


Ad-Hoc Network Mobility Emulation - /home/pbasu/mobiemu-1.2/master/scen-600x400-5-0-10
OpenFile
Engage
Pause
Reset
Exit
Config
Help
© 2001, HRL Laboratories, LLC
Dr. Yongguang Zhang (ygz@hrl.com)
HRL LABORATORIES
1 [out]
in[3]
0
in[2]
in[4]

Preview Mode? Show Links? [✓] Show Out-of-Range Links?
Emulation Speed: 1 Smoothness: 5

Status: Node 3: READY 192.1.1.103 00:07:0e:b4:26:9b Area: 600x400 Length: 300.0 sec Clock: 144.52

Figure 3: Screen-shot of MobiEmu Master Screen

4 Related Research Issues

Some other issues that will be highlighted during the demo include the following:

Reliability of Task Execution: If an instantiated pollee or a polling manager become uninstantiated at a certain time instant, ADUs in flight will be lost. In order to ensure delivery of those ADUs to newly re-instantiated nodes, a task-level reliability mechanism is necessary. We have implemented such a mechanism on top of UDP – in our scheme ADUs are buffered in the *taskd* processes of intermediate instantiated nodes. These are retransmitted if no acknowledgment is received from instantiated children nodes. The buffered ADUs are purged upon the expiration of a application-defined timer or after the arrival of an explicit `PURGE` message propagated from sink (packet destination).

Mobile Code: If the application process (*taskapp*) does not exist on a service providing (instantiated) device, its parent node is responsible for sending it the necessary code. The child's *taskd* process can then extract the code from the received buffer to a file which can then be spawned and executed as a separate process (see Figure 1). This process then acts as the *taskapp* on the child device. In addition, through the *Data-Flow Tuples*, a user may provide a program (binaries or scripts) that other nodes must execute. Such code is sent by the user task daemon (*taskd*) together with the tuples to the selected instances of the TG nodes in the MANET. Due to the necessity of code distribution, reliable transmission is actively supported in the TG framework, including packet buffering, retransmission and fragmentation and reassembly of large data chunks. Security issues however (authentication, code integrity and privacy concerns) are not yet supported at this stage and may be part of future work.

Possible Future Enhancements: Our goals for the current testbed extend beyond the polling application described in this paper. We are currently working on adding other features to the *taskapp* layer where more complex applications can be developed [1], e.g. a smart presentation application involving the adaptive selection of peripherals. Other future plans include carrying out extensive experiments to quantify performance with respect to the metrics mentioned in this paper under various conditions and comparing our reliability hooks in the UDP transport protocol to different versions of the TCP protocol.

References

[1] P. Basu. *A Task Based Approach for Modeling Distributed Applications on Mobile Ad Hoc Networks*. PhD thesis, Boston University, May 2003. `http://hulk.bu.edu/projects/pbasu-PhDthesis2003.pdf`.

[2] P. Basu, W. Ke, and T. D. C. Little. "A New Task-Based Approach for Supporting Distributed Applications on Mobile Ad Hoc Networks". *Computer Communication Review*, 32(1):63, January 2002. published as a abstract of a poster that appeared in ACM SIGCOMM 2001.

[3] P. Basu, W. Ke, and T. D. C. Little. "A Novel Approach for Execution of Distributed Tasks on Mobile Ad Hoc Networks". In *Proceedings of the IEEE Wireless Computing and Networking Conference (WCNC)*, Orlando, FL, March 2002.

[4] P. Basu, W. Ke, and T. D. C. Little. "Dynamic Task Based Anycasting in Mobile Ad Hoc Networks". *ACM/Kluwer Journal for Mobile Networks and Applications (MONET)*, 8(5), October 2003. To Appear.

[5] OLSR Implementation for Linux: INRIA, France. URL. `http://menetou.inria.fr/olsr/`.

[6] Y. Zhang and W. Li. "An Integrated Environment for Testing Mobile Ad-Hoc Networks". In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, Switzerland, June 2002.